UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/684,053	10/09/2003	Chandan Mathur	1934-12-3	3240

7590 06/08/2010

Bryan A. Santarelli
GRAYBEAL JACKSON HALEY LLP
Suite 350
155-108th Avenue NE
Bellevue, WA 98004-5901

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
06/08/2010	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.** 10/684,053	**Applicant(s)** MATHUR ET AL.	
	Examiner DAVID J. HUISMAN	**Art Unit** 2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *28 May 2010*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-15 and 17-61* is/are pending in the application.

4a) Of the above claim(s) *25-36 and 55-61* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-15,17-24 and 37-54* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 May 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *4/19/10*.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-15 and 17-61 are pending. Claims 25-36 and 55-61 have been withdrawn. Claims 1-15, 17-24, and 37-54 have been examined.

Continued Examination Under 37 CFR 1.114

2. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on May 28, 2010, has been entered.

Information Disclosure Statement

3. Applicant's IDS filed on April 19, 2010, fails to fully comply with 37 CFR 1.98(b)(5). Consequently, the following documents have been struck through and not considered:

- NPL document 1, for not citing the author.
- NPL document 2, for not citing the title or the month of publication. Note from MPEP 609.04(a), "The date of publication supplied must include at least the month and year of publication, except that the year of publication (without the month) will be accepted if the applicant points out in the information disclosure statement that the year of publication is sufficiently earlier than the effective U.S. filing date and any foreign priority date so that the particular month of publication is not in issue."

- NPL document 3, for not citing the relevant pages. For instance, the citation should include "2 pages".
- NPL document 4, for not citing the relevant pages. For instance, the citation should include "2 pages".
- NPL document 5, for not citing the relevant page numbers. For instance, the citation should include “4 pages”. Also, please insert a comma between both instances of “Office”.
- NPL document 6 for not citing the month of publication. Again, note from MPEP 609.04(a), “The date of publication supplied <u>must include at least the month and year of publication</u>, except that the year of publication (without the month) will be accepted <u>if the applicant points out in the information disclosure statement that the year of publication is sufficiently earlier than the effective U.S. filing date and any foreign priority date so that the particular month of publication is not in issue</u>.”

Specification

4. The amended title of the invention submitted by applicant on December 27, 2007, is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed. The examiner asserts that many prior art systems exist which include “buffered data transfer”. The examiner requests that applicant amend the title to include language directed towards the novelty of the claimed invention. MPEP 606.01 states that such an amendment “may result in slightly longer titles, but the loss in brevity of title will be more than offset by the

gain in its informative value in indexing, classifying, searching, etc. If a satisfactory title is not supplied by the applicant, the examiner may, at the time of allowance, change the title by examiner's amendment." Through the examiner doesn't have any recommendations at this moment in time, the examiner would like to give applicant every opportunity to submit an informative title.

Claim Objections

5. Claim 10 is objected to because of the following informalities: In line 3, it appears that the comma after "to" should be replaced with a colon.

6. Claim 13 is objected to because of the following informalities: Please insert --to-- between "operable" and the colon in line 2, and delete "to" from the beginning of each of lines 3, 4, and 5.

7. Claim 19 is objected to because of the following informalities: In line 4, it appears that the comma after "to" should be replaced with a colon.

8. Claim 21 is objected to because of the following informalities:

- In line 4, replace the comma after "to" with a colon.
- In line 6, delete "to".

9. Claim 22 is objected to because of the following informalities:

- In line 8, please replace the comma after "to" with a colon.
- In the 3rd to last paragraph, insert a comma after "buffer" and after "object".

10. Claim 23 is objected to because of the following informalities:

- In line 2, delete the comma after the colon and insert --to-- between “operable” and the colon.
- In lines 3 and 5, delete “to” from the beginning of the lines.

11. Claim 37 is objected to because of the following informalities: In line 2, insert a comma after “publishing” and after “application”.

12. Claim 45 is objected to because of the following informalities: In line 5, insert a comma after “buffer” and after “object”.

13. Claim 53 is objected to because of the following informalities:

- In lines 2-3, insert a comma after “generating” and after “instruction”.
- In lines 5-6, insert a comma after “generating” and after “instruction”.
- In the 3rd to last paragraph, insert a comma after “buffer” and after “processor”.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

14. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

15. Claims 10-15, 17-18, 37-50 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the enablement requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to enable one skilled in the art to which it pertains, or with which it is most nearly connected, to make and/or use the invention.

16. Referring to claims 10, 37, and 45, it is not clear to the examiner how to make/use the invention without generating data destination information (or the like). With respect to applicant's Fig.5, a given thread (for instance, thread 100_3) may publish data to multiple buffers 106 by way of multiple data transfer objects 86. It is not clear how the thread specifies which N buffers are to be written to without generating an address to identify them. It appears that some sort of addressing must occur to specify the destination(s) of the data. Otherwise, the system would have to be hardwired and static, with the same buffers always being written to by the same threads. Clarification is requested.

17. Claims 11-15, 17-18, 38-44, and 46-50 are rejected under 35 USC 112, 1st paragraph, for being dependent on a claim lacking enablement.

Claim Rejections - 35 USC § 102

18. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

19. Claims 10-12, 14-15, 17-18, 37, 39-43, 45, and 47-49 are rejected under 35 U.S.C. 102(b) as being anticipated by Dretzka et al., U.S. Patent No. 4,703,475 (herein referred to as Dretzka). Note that some of the claims are rejected multiple times under different interpretations of Dretzka.

20. Referring to claim 10, Dretzka has taught a computing machine, comprising:

a) a first buffer. See Fig.6, buffer 220-4, for instance.

b) a processor (Fig.1, component 21) coupled to the buffer and operable to:

b1) execute first and second data-transfer objects and an application. Fig.4 sets forth at least some of the data-transfer objects executed by processor 21. Looking at Fig.4 and Fig.6, a first transfer object would be executed to load data into buffer 220-4 and a second object would be executed to unload data from buffer 220-4. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

b2) generate data under control of the application such that the generated data includes no data-destination information. See Fig.6 and Figs.8-15, and note that the level 2.5 application generates data using an input list by searching for consecutive sequence packets. Also, note from Fig.4, at level 2.5, the 1-byte header (data-destination information) is deleted, and therefore, the data includes no data-destination information. In an alternate interpretation, if an address is inherently generated in Dretzka, as applicant argues, the address may still be considered as being separate from the generated data (i.e., the true content of the message). Therefore, the generated data itself does not include any data-destination information.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer object. See Fig.6. After data is generated by the input list, the first object transfers it to the buffer (i.e., 220-4).

b3) unload the data from the buffer under the control of the second data-transfer object. See Fig.6. Data is ultimately moved/unloaded from the buffer 220-4 and into buffer 210 under control of the second object.

b4) process the unloaded data under the control of the application such that the processed data includes only non-data-destination information. See Fig.4 and Fig.6. From the buffer 210, the processor will process the data using an application. Also, note from Fig.4, at level 2.5 (which occurs well before the unloading), the 1-byte header (data-destination information) is deleted, and therefore, the processed data includes no data-destination information. In the alternate interpretation, if an address inherently exists, as argued by applicant, then this address is still separate from the generated data. Hence, the generated data does not include data-destination information.

21. Referring to claim 11, Dretzka has taught the computing machine of claim 10 wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

22. Referring to claim 12, Dretzka has taught the computing machine of claim 10 wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.4) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the

first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

23. Referring to claim 14, Dretzka has taught the computing machine of claim 10 wherein the processor is further operable to:

a) execute a queue object and a reader object. See Fig.4 and Fig.6. The queue object is the object which stores the more bit, which ultimately leads to the informing of level 4 that a complete message has been received. The reader object is the object which detects the more bit so that further action can occur.

b) store a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. See Fig.4 and Fig.6. The queue object will store the "more" bit, which reflects the loading of the retrieved data into the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

c) read the queue value under the control of the reader object. See Fig.4 and note that the more bit, when set to a certain level, will indicate the end of the message and that the message may be further transmitted and processed. The reader object will have to read this more bit.

d) notify the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the "more" bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer. See Fig.4 and Fig.6.

e) unload the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

24. Referring to claim 15, Dretzka has taught the computing machine of claim 10, further comprising:

a) a second buffer. See Fig.6, component 210.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.4 and Fig.6. Data is unloaded from buffer 220-4 to buffer 210 under control of the second transfer object, and then moved from buffer 210 to the application in the processor under control of the third object.

25. Referring to claim 17, Dretzka has taught the computing machine of claim 10 wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

b) the processor is operable to execute an object factory and to generate the object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for

data to be received, the first and second object codes will be generated and invoked so that data may be received.

26. Referring to claim 10, Dretzka has taught a computing machine (under a second interpretation), comprising:

a) a first buffer. See Fig.5, buffer 110, for instance.

b) a processor (Fig.1, component 11) coupled to the buffer and operable to:

> b1) execute first and second data-transfer objects and an application. Fig.3 sets forth at least some of the data-transfer objects executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer object would be executed to load data into buffer 110 and a second object would be executed to unload data from buffer 110 and into buffer 120-4, for instance. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.
>
> b2) generate data under control of the application such that the generated data includes no data-destination information. See Fig.5 and note that a message comprising data must inherently be generated before being stored in buffer 110. This data, as shown at the top of Fig.3, is generated as a result of application execution. Note that at this time of generation, no headers and/or data-destination information are attached to the generated data. In an alternate interpretation, if an address is inherently generated in Dretzka, as applicant argues, the address is separate from generated data. Therefore, the generated data itself does not include any data-destination information.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer object. See Fig.5. After data is generated, it is ultimately stored in buffer 110.

b3) unload the data from the buffer under the control of the second data-transfer object. See Fig.5. Data is ultimately moved/unloaded from the buffer 110 and into buffer 120-4 under control of the second object.

b4) process the unloaded data under the control of the application such that the processed data includes no data-destination information. See Fig.3 and note that the data, after being moved into buffer 120-4, is further processed by breaking the message up. It isn't until the data is in the next buffer that a 1-byte channel header, which may or may not be considered data-destination information is added to it. Hence, at the time of the claimed processing, since the 1-byte header has not been added, the data does not include data-destination information. In the alternate interpretation, if an address inherently exists, as argued by applicant, then this address is still separate from the generated data. Hence, the generated data does not include data-destination information.

27. Referring to claim 11, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a source and stores it in a next buffer 110. Likewise, the second object retrieves data from a source (buffer 110) and stores it in a next buffer 120-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

28. Referring to claim 12, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.3) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

29. Referring to claim 15, Dretzka has taught the computing machine of claim 10 (under a second interpretation), further comprising:

a) a second buffer. See Fig.5, component 120-4.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.3 and Fig.5. Data is unloaded from buffer 110 to buffer 120-4 (second buffer) under control of the second transfer object, and then moved from buffer 120-4 to the an additional buffer and interface under control of the third object.

30. Referring to claim 17, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a source and stores it in a next buffer 110. Likewise, the second object retrieves data from a source (buffer 110) and stores it in a next buffer 120-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

b) the processor is operable to execute an object factory and to generate the object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

31. Referring to claim 18, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein the processor is further operable to package the generated data into a message that includes a header and the data under the control of the second data-transfer object. See Fig.3 (at least the level 3 object code), and note that the second object begins packaging the data into a message.

32. Referring to claim 37, Dretzka has taught a method comprising:

a) publishing with an application data that includes no information indicating a destination of the data. See the top of Fig.3 and note that the processor produces (publishes) data messages as a result of application execution. Note that, at the time of generation, no destination information is produced. In an alternate interpretation, if an address is inherently generated in Dretzka, as

applicant argues, the address is separate from generated data. Therefore, the generated data itself does not include any data-destination information.

b) loading the published data into a first buffer with a first data-transfer object, the loaded data including no information indicating a destination of the data, each location within the buffer corresponding to a same data destination. See Fig.3 and Fig.5 and note that the published data may be written to buffer 120-4 under control of the first transfer object (the code which performs the loading). Each location within the buffer corresponds to the same data destination, i.e., the buffer comprising the locations. For purposes of examination, data-destination information <u>may be</u> interpreted as the 1-byte multi-link header information added in level 2.5. Therefore, when the data is loaded into buffer 120-4 at level 3, the data does not include that header, and therefore, does not include information indicating a destination of the data.

c) retrieving the published data from the buffer with a second data-transfer object, the retrieved data including no information indicating a destination of the data. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The published data is retrieved from the first buffer under the control of the second transfer object (the code which performs at least the retrieving). Again, when the data is retrieved it still does not contain the information indicating the data's destination. This information isn't added until the data is stored in the buffer in level 2.5. In the alternate interpretation, if an address inherently exists, as argued by applicant, then this address is still separate from the generated data. Hence, the generated data does not include data-destination information.

d) generating a message header that includes a destination of the retrieved data. See Fig.3 and note the code performed at level 2.5. In this step, a message header including logical channel number destination is generated and attached to the data.

e) generating a message that includes the retrieved data and the message header. See Fig.3 and note that a 22-byte message is produced from the 21-byte data and 1-byte header.

33. Referring to claim 39, Dretzka has taught the method of claim 37, further comprising:

a) generating a queue value that corresponds to the presence of the published data in the buffer. See Fig.3 and note that when the last part of the message is found, it is indicated by generating a "more" bit (queue value).

b) notifying the second data-transfer object that the published data occupies the buffer in response to the queue value. Note from Fig.3 that when the last packet is found and the more bit is set, the code which then retrieves the data must be notified.

c) wherein retrieving the published data comprises retrieving the published data from the buffer with the second data-transfer object in response to the notification. See Fig.3, and note the level 2.5 code.

34. Referring to claim 40, Dretzka has taught the method of claim 37, further comprising driving the message onto a bus with a communication object. See Fig.3 and Fig.5. After being stored in the level 2.5 buffer, the message is driven on the bus with a communication object.

35. Referring to claim 41, Dretzka has taught the method of claim 37, further comprising loading the retrieved data into a second buffer with the second data-transfer object. See Fig.3 and Fig.5 and note that after being stored in a first buffer 120-4, the second object retrieves the data and stores it in another buffer 130-4, for instance.

36. Referring to claim 42, Dretzka has taught the method of claim 37 wherein generating the message header and the message comprise generating the message header and the message with the second data transfer object. See Fig.3, level 2.5 code, in which the header is generated and attached to the data.

37. Referring to claim 43, Dretzka has taught the method of claim 37, further comprising:

a) generating data-transfer object code with an object factory. All processors execute programs. The program (object factory) will dictate/generate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted or received, the data transfer objects will be invoked so that data may be transmitted and received. Clearly, the hardware shown in the figures must have some software interaction because without software, the hardware would be useless.

b) generating the first data-transfer object as a first instance of the object code. When data needs to be transferred, some type of “send” or “pass” or “store” command must be generated. This command is part of the data transfer object.

c) generating the second data-transfer object as a second instance of the object code. Similarly, when data needs to be transferred from first buffer to second buffer for header and message generation, some type of command needs to be generated. This command is part of the second data transfer object.

38. Referring to claim 45, Dretzka has taught a method comprising:

a) receiving a message that includes data and that includes a message header that indicates a destination of the data, the destination corresponding to a software application. See Fig.4 and Fig.6. Note that a message comes in with a 1-byte header (note level 2.5) that will indicate, upon

review by the receiving end, a destination of either input list 230-x or message buffer 220-x (column 7, line 25, to column 8, line 31). Clearly, the message must be received by some corresponding application.

b) loading into a first buffer with a first data-transfer object, the received data without the message header, the first buffer corresponding to the destination. See Fig.6 and note that the data is loaded into buffer 220-4 based on the 1-byte header. Note that the 1-byte message header is removed prior to storage in 220-4 according to Fig.4.

c) unloading the data from the buffer with a second data-transfer object. See Fig.4 and Fig.6 and note that data is ultimately unloaded from buffer 220-4.

d) processing the unloaded data with an application corresponding to the destination. See Fig.4 and note that after the data is unloaded, it will be sent to the processor where inherent processing on that data will commence.

39. Referring to claim 47, Dretzka has taught the method of claim 45, further comprising:

a) generating a queue value that corresponds to the presence of the data in the buffer. See Fig.4 and Fig.6. The queue object will generates the “more” bit, which corresponds to the presence of data in the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

b) notifying the second data-transfer object that the data occupies the buffer in response to the queue value. When the “more” bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer by informing (notifying) the next level that data is ready to be transferred. See Fig.4 and Fig.6.

c) wherein unloading the data comprises unloading the data from the buffer with the first data-transfer object in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

40. Referring to claim 48, Dretzka has taught the method of claim 45, further comprising wherein receiving the message comprises receiving the message with the first data-transfer object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data. That object is the first data transfer object.

41. Referring to claim 49, has taught the method of claim 45, further comprising:

a) receiving the message comprises retrieving the message from a bus with a communication object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data from a bus. That object is the communication object.

b) transferring the data from the communication object to the first data transfer object. See Fig.4 and Fig.6. Note that after the data is received, it is passed to the first data transfer object which at least stores it in buffer 220-4.

Claim Rejections - 35 USC § 103

42. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

43. Claims 1-3 and 5-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka in view of Chamdani et al., U.S. Patent No. 6,985,975 (herein referred to as Chamdani).

44. Referring to claim 1, Dretzka has taught a computing machine, comprising:

a) first and second parallel buffers (see Fig.5, buffers 120-0 and 120-4, for instance) respectively associated with first and second data-manipulation units (each buffer inherently has a respective input wire/bus, i.e., data-manipulation unit. Note that a data-manipulation unit is broadly interpreted as any unit in a data-manipulation system, which Dretzka has taught. "data-manipulation" is merely a non-functional label/name for a unit).

b) a processor (Fig.1, component 11) coupled to the buffers and operable to:

> b1) execute an application and first and third data-transfer objects. All processors inherently execute an application. And, Fig.3 sets forth the data-transfer objects executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer object would be executed to load data into buffer 120-0 and a third object would be executed to retrieve data from buffer 120-0 for loading into buffer 130-0.
>
> b2) publish data under the control of the application. See the top of Fig.3 and note that the processor produces (publishes) data messages as a result of application execution.
>
> b3) load at least a portion of the published data into the first buffer under the control of the first data-transfer object. See Fig.3 and Fig.5 and note that the published data may be written to message buffers 120-0 under control of the first data transfer object.
>
> b4) retrieve at least the portion of the published data from the first buffer under the control of the third data-transfer object. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The published data is retrieved from the first buffer under the control of the first transfer object.

b5) Dretzka has not taught that the processor is operable to execute second and fourth data-transfer objects, load at least the same portion of the published data into the second buffer under control of the second data-transfer object, and retrieve at least the portion of the published data from the second buffer under the control of the fourth data transfer object. However, Chamdani has taught the concept of redundant transfer of messages throughout a system. See the abstract, Fig.2, Fig.5, and claim 1. Essentially, a message is loaded into and retrieved from a first buffer. The same message is loaded into and retrieved from a second buffer. The messages from the first and second buffer are then compared to ensure that they are the same, thereby indicating that a reliable transfer has occurred. Consequently, in order to implement reliable transfer within Dretzka, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the transferring components of Dretzka are essentially replicated. That is, currently in Dretzka, first and third objects load and retrieve a message from a first buffer. Modifying Dretzka in view of Chamdani would result in also having second and fourth objects that load and retrieve the same message from a second buffer. These messages would then be compared to ensure reliability.

45. Referring to claim 2, Dretzka in view of Chamdani has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. See Fig.5 and note that, in general, the first object retrieves data from a previous buffer 110 and stores it in a next buffer 120-0 in channel 0. Likewise, the third object retrieves

data from a previous buffer 120-0 and stores it in a next buffer 130-0 in channel 0. Hence, both of these objects comprise instances of general "channel 0 retrieve-and-store" object code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. In Dretzka, as modified in view of Chamdani, the second object loads a redundant buffer. Likewise, the fourth object retrieves data from the redundant buffer. Hence, both of these objects comprise instances of general "redundant" object code.

46. Referring to claim 3, Dretzka in view of Chamdani has taught the computing machine of claim 1 wherein the processor comprises:

a) a processing unit operable to execute the application and publish the data under the control of the application. Recall from the rejection of claim 1 (and from Fig.3) that the processor inherently executes an application and publishes data under control of the application. This execution and publishing is performed by a processing unit.

b) a data-transfer handler operable to execute the first, second, third, and fourth data-transfer objects, to load the published data into the first and second buffers under the control of the first and second data-transfer objects, respectively, and to retrieve the published data from the first and second buffers under the control of the third and fourth data-transfer objects, respectively. Again, recall from the rejection of claim 1 that first and second objects loading the first and redundant buffers, respectively, and third and fourth objects retrieve the data from first and redundant buffers, respectively. The unit which performs this execution for loading and retrieving data is a data transfer handler.

47. Referring to claim 5, Dretzka in view of Chamdani has taught the computing machine of claim 1 wherein the processor is further operable to:

a) execute a queue object and a reader object. See Fig.3 and Fig.5. The queue object is the object which causes data from buffer 110 to at least be broken up, attached to a header, and have a “more” bit set, before being stored in buffers in level 3. The reader object is the object which at least reads the “more” bit to cause further action to occur.

b) store a queue value under the control of the queue object, the queue value reflecting the loading of the published data into the first buffer. See Fig.3 and Fig.5. Values are written into the level 3 storage until the "more" bit is set low, which means loading of the data is done.

c) read the queue value under the control of the reader object. The “more” bit would not be set for no reason. It clearly serves a purpose, and it will be read. The object which reads it is the reader object.

d) notify the third data-transfer object that the published data occupies the first buffer under the control of the reader object and in response to the queue value. When the “more” bit is set and noted by the reader object, the data may be transferred to the next level of buffers. See Fig.3 and Fig.5.

e) retrieve the published data from the first buffer under the control of the third data-transfer object and in response to the notification. Again, in response to the third object, the data will be moved from level 3 buffers to level 2.5 buffers.

48. Referring to claim 6, Dretzka in view of Chamdani has taught the computing machine of claim 1, further comprising:

a) a bus. See Fig.2, at least components 40-4, 40-3, etc.

b) wherein the processor is operable to execute a communication object and to drive the data retrieved from one of the first and second buffers onto the bus under the control of the

communication object. See Fig.2 and Fig.5. Note that after data is stored in a level 2.5 buffer (second buffers), a communication object will ultimately drive that data onto the bus via a digital facility interface (DFI) 14-0, 14-1, etc.

49. Referring to claim 7, Dretzka in view of Chamdani has taught the computing machine of claim 1, further comprising:

a) a third buffer. See Fig.5, component 130-0, for instance.

b) wherein the processor is operable to provide the data retrieved from one of the first and second buffers to the third buffer under the control of the respective one of the third and fourth data-transfer objects. Recall from Fig.5 that the first buffer is buffer 120-0 and that data will ultimately be moved from buffer 120-0 to buffer 130-0. The third object is the object which moves the data between these two buffers the first and third buffer.

50. Referring to claim 8, Dretzka in view of Chamdani has taught the computing machine of claim 1 wherein the processor is further operable to generate a message that includes a header and data retrieved from one of the first and second buffers under the control of the respective one of the third and fourth data-transfer objects. See Fig.3 (at least the level 2.5 object code), and note that the third object removes the data from buffers in level 3 and adds a header to the data to form a 22-byte packet.

51. Referring to claim 9, Dretzka in view of Chamdani has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. See Fig.5 and note that, in general, the first object retrieves data from a previous buffer 110 and stores it in a next buffer 120-0 in channel 0. Likewise, the third object retrieves

data from a previous buffer 120-0 and stores it in a next buffer 130-0 in channel 0. Hence, both of these objects comprise instances of general "channel 0 retrieve-and-store" object code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. In Dretzka, as modified in view of Chamdani, the second object loads a redundant buffer. Likewise, the fourth object retrieves data from the redundant buffer. Hence, both of these objects comprise instances of general "redundant" object code.

c) the processor is operable to execute an object factory and to generate the first object code and the second object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

52. Claim 4 is rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka in view of Chamdani, and further in view of the examiner's taking of Official Notice.

53. Referring to claim 4, Dretzka in view of Chamdani has taught the computing machine of claim 1. Dretzka has not taught that the processor is further operable to execute a thread of the application and to publish the data under the control of the thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, the system may switch to a second thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it

would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor executes a thread of the application and publishes data under control of the thread.

54. Claims 13-14, 19-24, 38, 44, 46, 50, and 53-54 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka in view of the examiner's taking of Official Notice.

55. Referring to claim 13, Dretzka has taught the computing machine of claim 10 (under both the first and second interpretations of Dretzka). Dretzka has not taught that the processor is further operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, and because generating data and processing unloaded data are independent tasks, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread.

56. Referring to claim 14, Dretzka has taught the computing machine of claim 10 (under a second interpretation).

a) Dretzka has not taught that the processor is further operable to execute a queue object and a reader object. However, recall from Fig.5 that data is initially stored in a queue. The examiner asserts that it is well known and advantageous to have an empty bit indicate the status of the queue because it prevents the queue from being read if it has no useful data in it, thereby saving time. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka to execute a queue object to set/clear an empty bit based on whether the queue contains valid data to be read, and also a reader object to read the empty bit such that the system knows when to read the valid data from the queue.

b) Dretzka, as modified, has further taught storing a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. This is deemed inherent as an empty bit would be stored.

c) Dretzka, as modified, has further taught reading the queue value under the control of the reader object. Again, this is deemed inherent because if an empty bit were implemented, it would be meant for reading so that the system knows when the queue is empty.

d) Dretzka, as modified, has further taught notifying the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the empty bit is clear and noted by the reader object, the data exists in the queue and should be handled.

e) Dretzka, as modified, has further taught unloading the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from buffer 110 to 120-4.

57. Referring to claim 19, Dretzka has taught a peer-vector machine comprising:

a) a buffer. See Fig.5, component 120-4.

b) a bus. See Fig.2, components 40-0, 40-1, etc.

c) a processor (Fig.1, components 11) coupled to the buffer and to the bus and operable to:

c1) execute an application, first and second data-transfer objects, and a communication object. And, Fig.3 sets forth at least some of the data-transfer objects executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer object would be executed to load data into buffer 120-0, a second object would be executed to load data into buffer 120-4, and third object would be executed to retrieve data from buffer 120-0 for loading into buffer 130-0, and a fourth object would be executed to retrieve data from buffer 120-4 for loading into buffer 130-4. At least some functions, other than those performed by the first and second objects, are performed by the communication object.

c2) publish data under the control of the application. See the top of Fig.3 and note that the processor produces (publishes) data messages as a result of application execution.

c3) load the published data into the buffer under the control of the first data-transfer object. See Fig.3 and Fig.5 and note that the published data may be written to message buffers 120-4, etc. over time, under control of the first transfer object.

c4) retrieve the published data from the buffer under the control of the second data-transfer object. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The

published data is retrieved from the first buffer under the control of the second transfer object.

c5) construct a message under the control of the second data-transfer object, the message including the retrieved published data and information indicating a destination of the retrieved published data. See Fig.3 and note that a 1-byte header including destination information (channel number) is attached to the data, thereby forming a message to send to the receiver.

c6) drive the published data onto the bus under the control of the communication object. See Fig.2 and Fig.5. Note that after data is stored in a level 2.5 buffer (second buffer), a communication object will ultimately drive that data onto the bus via a digital facility interface (DFI) 14-0, 14-1, etc.

d) a pipeline accelerator (Fig.6) coupled to the bus, including the destination (logical channel specified by the 1-byte header), and operable to receive the message from the bus, to recover the received published data from the message (Fig.4, deletion of the 1-byte header leaves at least the published data), to provide the recovered data to the destination (to the logical channel), and to process the recovered data at the destination (the data is processed at the levels at the destination as indicated in Fig.4, and also as argued by applicant in the affidavit filed on May 28, 2010). Note that "pipeline accelerator" is just the name of the unit coupled to the bus. Also, it may be considered a pipeline accelerator because the interface logic is inherently designed to accelerate data between inherently pipelined processors as fast as possible (given other constraints on size and power, if they exist).

e) Dretzka has not explicitly taught that the recovered data is processed at the destination without executing a program instruction. However, as is known in the art, and even as admitted by applicant in the affidavit filed on May 28, 2010, in paragraph [23], network interface logic may be implemented in hardware or software, depending on design choice, where hardware is generally faster. One known hardware implementation is on an FPGA, which is a well known and advantageous component that allows reactive logic to be programmed on chip to produce a desired function (with no instruction execution). An FPGA may be efficiently reprogrammed by a designer after manufacture, thereby realizing specialized circuitry for processing in particular environments. The reprogrammability also allows a designer to upgrade, completely change, or otherwise modify a configuration as needed for increased efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the network interface functionality is implemented on an FPGA, which in turn processes recovered data using programmed logic instead of instruction execution.

58. Referring to claim 20, Dretzka, as modified, has taught the peer vector machine of claim 19, wherein the destination includes a field-programmable gate array that is operable to process the recovered data. See the rejection of claim 19.

59. Referring to claim 21, Dretzka, as modified, has taught the peer vector machine of claim 19, further comprising

a) a registry coupled to the processor and operable to store object data. The examiner asserts that the processor's objects are made up of instructions which must be stored somewhere so that the processor may access and execute them. The storage holding the instructions is the registry.

b) wherein the processor is operable to execute an object factory, and to generate the first and second data-transfer objects and the communication object from the object data under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

60. Referring to claim 22, Dretzka has taught a peer vector machine comprising:

a) a buffer. See Fig.6, component 220-4, for instance.

b) a bus. See Fig.2, component 40-1, 40-2, etc.

c) a pipeline accelerator coupled to the bus and operable to generate data, to generate a header including information indicating a destination of the data, to package the data and header into a message, and to drive the message onto the bus. See Fig.1, unit 11, and Fig.3 and Fig.5. Note that the unit generates data, and then a message including the data and a header (with destination data specifying logical channel), and then drives that data onto bus 40-0, 40-1, etc (Fig.2), through the digital facility interface. Note that "pipeline accelerator" is just the name of the unit coupled to the bus. Also, it may be considered a pipeline accelerator because the interface logic is inherently designed to accelerate data between inherently pipelined processors as fast as possible (given other constraints on size and power, if they exist).

d) a processor (Fig.1, component 21) coupled to the buffer and to the bus and operable to:

> d1) execute an application, first and second data-transfer objects, and a communication object. Fig.4 sets forth at least some of the data-transfer objects executed by processor 21. Looking at Fig.4 and Fig.6, a first transfer object would be executed to load data into

buffer 220-4 and a second object would be executed to unload data from buffer 220-4 and into buffer 210. The communication object would be executed to retrieve data from the DFI 14-0, 14-1, etc, and store it into one of buffers 230-4. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

d2) receive the message from the bus under the control of the communication object. See Fig.4 and Fig.6. Note that a message is received from the interface under the control of the communication object.

d3) load into the buffer under the control of the first data-transfer object the received data without the header, the buffer corresponding to the destination of the data. See Fig.4 and Fig.6 and note that before being stored in the first buffer in level 3, the 1-byte header is deleted.

d4) unload the data from the buffer under the control of the second data-transfer object. See Fig.4 and Fig.6 and note that from the first buffer, the data is moved to the buffer 210 before ultimately being sent to the processor 21.

d5) process the unloaded data under the control of the application. See Fig.4 and Fig.6. From the buffer 210, the processor will process the data using an application.

e) Dretzka has not taught that the pipeline accelerator generates data without executing a program instruction. However, as is known in the art, and even as admitted by applicant in the affidavit filed on May 28, 2010, in paragraph [23], network interface logic may be implemented in hardware or software, depending on design choice, where hardware is generally faster. One known hardware implementation is on an FPGA, which is a well known and advantageous

component that allows reactive logic to be programmed on chip to produce a desired function (with no instruction execution). An FPGA may be efficiently reprogrammed by a designer after manufacture, thereby realizing specialized circuitry for processing in particular environments. The reprogrammability also allows a designer to upgrade, completely change, or otherwise modify a configuration as needed for increased efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the network interface functionality is implemented on an FPGA, which in turn generates data using programmed logic instead of instruction execution.

61. Referring to claim 23, Dretzka, as modified, has taught the peer vector machine of claim 22 wherein the processor is operable to receive the message from the bus under the control of the communication object, and to recover the data from the message under the control of the first data-transfer object. See Fig.4 and note that the communication object receives the message from the bus. And, the first data object recovers the data from the message. See at least the level 3 code of Fig.4.

62. Referring to claim 24, Dretzka, as modified, has taught the peer vector machine of claim 22, further comprising:

a) a registry coupled to the processor and operable to store object data. The examiner asserts that the processor's objects are made up of instructions which must be stored somewhere so that the processor may access and execute them. The storage holding the instructions is the registry.

b) wherein the processor is operable to execute an object factory, and to generate the first and second data-transfer objects and the communication object from the object data under the control of the object factory. All processors execute programs. The program (object factory) will

dictate when data needs to be transmitted and received. That is, when the program calls for data to be received, the first and second object codes (and communication object codes) will be generated and invoked so that data may be received.

63. Referring to claim 38, Dretzka has taught the method of claim 37. Dretzka has not taught that publishing the data comprises publishing the data with a thread of the application. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that publishing the data comprises publishing the data with a thread of the application.

64. Referring to claim 44, Dretzka has taught the method of claim 37. Dretzka has further taught receiving the message and processing the data in the message with a processor (Fig.4 and Fig.6). Dretzka has not explicitly taught the receiving processor is a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's processor (Fig.1, component 21) such that it includes a hardwired pipeline for accelerating execution.

65. Referring to claim 46, Dretzka has taught the method of claim 45. Dretzka has not taught that processing the unloaded data comprises processing the unloaded data with a thread of the application corresponding to the destination. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, the system may switch to a second thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor processes the unloaded data with a thread.

66. Referring to claim 50, Dretzka has taught the method of claim 45. Dretzka has not explicitly taught generating the message header and the message with a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's processor (Fig.1, component 11) such that it includes a hardwired pipeline for accelerating execution. Note that in this series of rejections, processor 11 is the unit which generates the messages according to Fig.3.

67. Referring to claim 53, Dretzka has taught a method comprising:

a) generating with a pipeline accelerator a message header that includes a destination of data, the destination identifying a software application for processing the data. See Fig.3 and note the

level 2.5 logic generates a 1-byte header including a logical channel destination. Note that "pipeline accelerator" is just the name of the unit that generates the header, and therefore, it is generally not given patentable weight. Also, at least this logic may be considered a pipeline accelerator because the interface logic is inherently designed to accelerate data between inherently pipelined processors as fast as possible (given other constraints on size and power, if they exist).

b) generating with the pipeline accelerator a message that includes a header and the data. See Fig.3 and note that in generating the 1-byte header, the unit forms a 22-byte message with 21-bytes of data.

c) driving the message onto a bus with the pipeline accelerator. See Fig.3 and Fig.5. Note that after the messages are generated, they are sent over the bus shown in Fig.2 using the digital facility interface (DFI).

d) receiving the message from the bus with a communication object running on a processor. See Fig.4 and Fig.6. Note that processor 21 of Fig.1 is receiving the message using code shown in Fig.4.

e) loading into a buffer with a first data-transfer object running on the processor the received data absent the header, the buffer being identified by the destination. See Fig.4 and note the level 3 object which stores data into the buffer in level 3 minus the header which was stripped in level 2.5 of Fig.4.

f) unloading the data from the buffer with a second data-transfer object running on the processor. See Fig.4 and Fig.6 and note that data from a buffer in level 3 is ultimately removed and moved on through the system.

g) processing the unloaded data with an application running on the processor and identified by the destination. See Fig.4 and Fig.6. Eventually, the unloaded data makes its way to the processor for inherent processing.

h) Dretzka has not taught that the message header and message are generated by the pipeline accelerator without executing a program instruction. However, as is known in the art, and even as admitted by applicant in the affidavit filed on May 28, 2010, in paragraph [23], network interface logic may be implemented in hardware or software, depending on design choice, where hardware is generally faster. One known hardware implementation is on an FPGA, which is a well known and advantageous component that allows reactive logic to be programmed on chip to produce a desired function (with no instruction execution). An FPGA may be efficiently reprogrammed by a designer after manufacture, thereby realizing specialized circuitry for processing in particular environments. The reprogrammability also allows a designer to upgrade, completely change, or otherwise modify a configuration as needed for increased efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that at least some of the network interface functionality is implemented on an FPGA, which in turn generates the message header and message using programmed logic instead of instruction execution.

68. Referring to claim 54, Dretzka, as modified, has taught a method as described in claim 53, further comprising recovering the data from the message with the first data transfer object. See Dretzka, Fig.4, and note that recovering the data from the message begins with the first data transfer object executing in level 2.5 of Fig.4.

69. Claims 51-52 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka in view of Gilson, U.S. Patent No. 5,361,373.

70. Referring to claim 51, Dretzka has taught a method comprising:

a) publishing data with an application running on a processor. See the top of Fig.3 and note that the processor produces (publishes) data as a result of application execution.

b) loading the published data into a buffer with a first data-transfer object running on the processor. See Fig.3 and Fig.5 and note that the published data may be written to buffer 120-4 under control of the first transfer object (the code which performs at least the loading).

c) retrieving the published data from the buffer with a second data-transfer object running on the processor. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The published data is retrieved from the first buffer under the control of the second transfer object (the code which performs at least the retrieving).

d) generating information that indicates a destination of the retrieved data. See Fig.3 and note the code performed at level 2.5. In this step, a message header including logical channel number destination is generated and attached to the data.

e) packaging the retrieved data and the information into a message. See Fig.3 and note that a 22-byte message is produced from the 21-byte data and 1-byte header.

f) driving the message onto a bus with a communication object running on the processor. See Fig.3 and Fig.5. After being stored in the level 2.5 buffer, the message is driven on the bus with a communication object.

g) receiving the message from the bus and processing the published data with a processor, which inherently includes a hardwired pipeline (since all modern processors are inherently pipelined).

See Fig.4 and Fig.6. Note that unit 20 of Fig.1 receives the messages sent by unit 10 and the received data is ultimately processed by the processor 21.

h) Dretzka has not taught that the processor w/ hardwired pipeline is part of a pipeline accelerator that includes a field-programmable gate array. However, Gilson has taught that a processor may be implemented on an FPGA. See Fig.1 and column 6, lines 43-60. Implementing a processor on an FPGA is advantageous because an FPGA may be efficiently reprogrammed by a designer after manufacture, thereby realizing specialized circuitry for processing in particular environments. The reprogrammability would allow a designer to upgrade, completely change, or otherwise modify a configuration as needed for increased efficiency. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the pipelined processor 21 is part of a pipeline accelerator that includes a field programmable gate array.

71. Referring to claim 52, Dretzka, as modified, has taught a method as described in claim 51, further comprising:

a) generating a message that includes a header and the published data with the second data-transfer object. See Fig.3 and note that in addition to receiving the data from the level 3 buffer, the second object also generates the header in the level 2.5 code of Fig.4.

b) driving the data onto the bus comprises driving the message onto the bus with the communication object. See Fig.3 and Fig.5. After being stored in the level 2.5 buffer, the message is driven on the bus with a communication object.

c) receiving and processing the published data comprises receiving the message and recovering the published data from the message with the pipeline accelerator. Again, see Fig.4 and Fig.6.

Response to Arguments

72. Applicant's arguments filed on May 28, 2010, have been fully considered.

73. In response to applicant's response, the examiner has eliminated rejections based on Britton. However, new rejections have been introduced in their place.

74. The 132 affidavit submitted by applicant has been reviewed. Much time was spent describing the OSI model and how Dretzka is in violation of that model. The examiner would like to point out that Dretzka does not appear to state anywhere that her system is based entirely on the OSI model. Clearly, Dretzka did not try to patent the OSI model, which she admitted was known prior to her invention (column 1, lines 20-24). Instead, she stated that communication protocols are frequently defined using a layered approach based, <u>at least in part</u>, on the OSI model (column 1, lines 18-24). Hence, the examiner is not certain that applicant can state that all known OSI layers must also be in Dretzka, and that Dretzka is in violation for not illustrating them. And, even if additional layers must inherently be present in Dretzka, which applicant argues, they are largely irrelevant with respect to examination of the claims, as just the illustrated layers were relied upon to reject the claims.

75. With respect to the argument of the rejection of claims 10 and 37 on pages 22-26 of the remarks, the examiner asserts that even if a destination address must be generated (to specify one of modules 10, 20, and 30 for communication in Fig.1), this address may still be considered as being separate from the generated data (e.g., the payload) and, therefore, the generated data does not include data destination information. That is, the message data and the address of the

component to which the message data will be sent are distinct. Furthermore, please note that the examiner still maintains his previous interpretation as well (regarding the 1 and 3-byte headers).

76. With respect to the argument of the rejection of claim 45 on pages 26-27 of the remarks, the examiner asserts that the claim is broader than applicant is arguing. Claim 45 requires "receiving a message that includes data and that includes a message header that indicates a destination of the data…and loading into a first buffer the received data without the message header". Based on column 7, line 25, to column 8, line 31, the 1-byte header may be interpreted as destination information since it determines, upon review by the receiving end, a destination of either input list 230-x or message buffer 220-x. Hence, the message data and this 1-byte header are received. However, from Fig.6, this 1-byte header is stripped from the message and the remaining message is stored into buffer 220-4, for instance. Hence, the received data is loaded into the buffer without the 1-byte header. Even if Dretzka also inherently generates an address of module 20 or 30 (Fig.1) for communication purposes, this doesn't prevent Dretzka from reading on claim 45 in the manner set forth above.

77. With respect to the argument of the rejection of claim 1, the examiner asserts that the amendment does not overcome the rejection. Applicant is simply changing the name of the unit, from "data processing unit" to "data manipulation unit". Broadly, these refer to any unit in a data processing/manipulation system, which Dretzka has taught. Even a wire/bus is a data manipulation unit in a data manipulation system. "Data manipulation" is simply a name/label for the unit. A name/label, in this situation, doesn't set forth any functionality performed by the unit itself. Therefore, it need not be interpreted any more narrowly than as a wire/bus.

78. With respect to the argument of the rejection of claim 4, applicant essentially argues that despite Dretzka having an application, it is not obvious to have that application comprise multiple threads. As previously stated, multithreading is very well known and advantageous in the art. It allows for less stalling during execution of instructions, thereby allowing quicker generation of messages. Applicant asserts that multithreading would decrease performance in an OSI model and, therefore, it would not be an obvious modification. Even if this were true, one cannot deny performance gains obtained from multithreading due to less stalling (this is why multithreading exists). Hence, to a given designer, that gain may outweigh any loss in communication performance. For multithreading to not be obvious, it would appear that multithreading would have to be incompatible with OSI or some other layered communication model. And, the examiner is not aware that such incompatibility is certain. Therefore, until such is proven, the examiner will assume that they are compatible and that multithreading will be useful in such a system for reasons set forth above.

79. With respect to the first argument of the rejection of claim 19 on page 31, respectively, the examiner has pointed out the buffer and the data transfer objects in Dretzka. The latter are inherently present as logic must exist to cause a header to be added to the message and to cause data to be moved to/from a buffer. The remaining arguments of the rejection of claim 19 with respect to Britton are moot in view of the new grounds of rejection, which no longer includes Britton.

80. With respect to the first argument of the rejection of claim 22 on page 34, the examiner has pointed out the buffer and the data transfer objects in Dretzka. The latter are inherently present as logic must exist to cause a header to be stripped from the message and to cause data to

be moved to/from a buffer. The remaining arguments of the rejection of claim 22 with respect to Britton are moot in view of the new grounds of rejection, which no longer includes Britton.

81. The first arguments of the rejections of claims 51 and 53 are not persuasive for the same reasons set forth in responses to the arguments of the rejections of claims 19 and 22 above. The remaining arguments of the rejections of claims 51 and 53 with respect to Britton are moot in view of the new grounds of rejection, which no longer includes Britton.

82. Finally, applicant telephoned the examiner on May 28, 2010, and requested another phone call from the examiner if every claim was not found to be allowable. However, given the examiner's schedule and workload, the nature of applicant's response (including a lengthy 132 declaration which had to be considered), and the fact that the examiner's response was due approximately one week after the amendment was filed, there was not enough time to further discuss the application. However, the examiner is more than willing to discuss the application again after applicant reviews this response. Applicant is encouraged to call the examiner if desired.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DAVID J. HUISMAN whose telephone number is (571)272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/David J. Huisman/
Primary Examiner, Art Unit 2183